COMPUTER SYSTEMS LABORATORY

STANFORD UNIVERSITY · STANFORD, CA 94305-2192

THEORETICAL RESULTS ON DISTRIBUTED PROCESSING WITH LIMITED COMPUTING RESOURCES

Hemant Kanakia and Fouad A. Tobagi

SEL Technical Report No. 85-273

April 1985

This work was supported by the Defense Advanced Research Projects Agency under Contract No. MDA 903-79-C-0201 and MDA 903-84-K-0249.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 85-273	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* THEORETICAL RESULTS ON DISTRIBUTED PROCESSING WITH LIMITED COMPUTING RESOURCES		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER 85-273
7. AUTHOR(s) Hemant Kanakia and Fouad A. Tobagi		8. CONTRACT OR GRANT NUMBER(s) MDA 903-79-C-0201 MDA 903-84-K-0249
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford Electronics Laboratories Stanford University Stanford, California 94305-2192		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Blvd., Arlington, VA 22209		12. REPORT DATE April 1985 13. NUMBER OF PAGES 38
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Resident Representative Office of Naval Research Durand 165 Stanford University, Stanford, CA 94305-2192		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A task is a set of related operations which can be performed on some input data. An algorithm is a collection of tasks with an underlying structure defined in terms of precedence relationships among the tasks. Two models of processing systems are considered. A distributed processing system consists of separate processors, each one dedicated to perform a single task, and permits pipelined processing of consecutive requests, as well as concurrent processing of tasks for a given request. A centralized processing system consists of a

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

single processor which performs all the tasks for a given request sequentially, and services the requests sequentially without pipelining. The performance of these two systems is compared in terms of average execution times, under the constraint that the capacity of the processor in the centralized system is equal to the sum of the capacities of individual processors in the distributed system.

Theoretical Results on Distributed Processing With Limited Computing Resources*

SEL Technical Report No. 85-273
April 1985

Hemant Kanakia and Fouad A. Tobagi
Computer Systems Laboratory
Department of Electrical Engineering
Stanford University, Stanford, CA 94305-2192

Abstract

A task is a set of related operations which can be performed on some input data. An algorithm is a collection of tasks with an underlying structure defined in terms of precedence relationships among the tasks. Two models of processing systems are considered. A distributed processing system consists of separate processors, each one dedicated to perform a single task, and permits pipelined processing of consecutive requests, as well as concurrent processing of tasks for a given request. A centralized processing system consists of a single processor which performs all the tasks for a given request sequentially, and services the requests sequentially without pipelining. The performance of these two systems is compared in terms of average execution times, under the constraint that the capacity of the processor in the centralized system is equal to the sum of the capacities of individual processors in the distributed system.

*This work was supported by the Defense Advanced Research Projects Agency under Contracts MDA 903-79-C-0201 and MDA 903-84-K-0249, monitored by the Office of Naval Research.

I. Introduction

A **task** is a set of related operations which can be performed on some input data. An **algorithm** is a collection of tasks which, once performed, will have accomplished a well defined goal. In the context of computing, for example, a task may be a function, a subroutine, or a process; an algorithm is a complete computer program. In the context of a manufacturing plant, a task may be milling operations of a part, or assembly operations for a sub-assembly of a product; an algorithm is the complete production of a product. As the examples indicate, the tasks constituting an algorithm are such that they may require communication of results or synchronization among them. This imposes a partial order relationship in the execution of the tasks for a request to execute the algorithm. We shall use the term **algorithm** to refer to a collection of tasks with an underlying **structure**. A major aspect of this structure is the precedence relationships (i.e., the partial order of execution) among the tasks. For a given request to execute an algorithm, all the tasks, as specified by the algorithm and its structure, are performed on some given input data.

We examine two models of processing systems which perform an algorithm for a stream of execution requests. In the **parallel execution** model, several processors are provided which run asynchronously. Each processor is dedicated to perform specific tasks. The system is such that different requests to execute the algorithm may be serviced at the same time at different processors in the system. This property is referred to as **pipelined processing** of requests. The system also allows many processors to simultaneously execute their tasks for a given execution request, as long as the restrictions imposed by the algorithm's task structure are satisfied. This property is referred to as **concurrent processing** of tasks. A processing system which follows the above model for servicing execution requests is referred to as a **distributed processing system**.

In the **sequential execution** model, a single processor is provided which can perform all tasks. In this case, the processor can service only one request at a time. When servicing

a request, the processor executes the tasks sequentially. We refer to such a system as a **centralised processing system.**

In this report, we compare the performance of a distributed processing system with that of a centralized processing system, performing an algorithm with a given task structure for a stream of execution requests, in terms of the average execution time for a request. Average execution time includes waiting time at the processors along with the actual time for the processors to perform the tasks. The performance comparison is done with the constraint that the total capacity of computing resources is the same for both processing systems. This constraint can be seen as an attempt to compare the performance of two systems with the same cost, where cost is considered to be proportional to the capacity of a processor. In this comparison, it is assumed that communications among processors are instantaneous and for free. There exist other important criteria by which to compare distributed and centralized processing systems, such as reliability, ease of system design, maintainability, flexibility in adapting to other algorithms, etc. Such criteria are not considered in this report.

Similar comparative studies have appeared in the literature for multi-server queues [1-4], multiple resource systems [4], and a restricted type of a distributed system [5]. Morse [1] was the first to consider an optimization problem for a multi-server queue. He considered an M/M/m queue with First-Come-First-Serve (FCFS) service discipline where each server services a job at rate C/m operations/sec. He found that, if the average waiting time is to be minimized, then m should be as high as possible; but if the average system delay is to be minimized, then m should be equal to one. Stidham [2] extended the above result to G/M/m, G/D/m, and G/E_k/m queues. Brumelle [3] considered a G/G/m queue and showed that, as long as the service time distribution has a coefficient of variation that is less than or equal to one, the average system delay is minimized when m is equal to one. Kleinrock [4], in addition to surveying these results, considered a system where each server

has its own queue of jobs. Assuming Poisson arrivals of jobs and exponential distributions for the service times, he shows that a single queue with a single server of the combined capacity is better than the multiple queue system.

The distributed system considered here is more general than the systems mentioned above in the sense that each request for the execution of the algorithm, will in general require service from more than one processor. There has been a recent paper by Kleinrock [5] which considers such a distributed system. The system analyzed in [5] is somewhat restricted, in the sense that only algorithms which consist of a series of tasks are considered. The model of precedence relationships developed by us is more general and contains the sequential precedence relationship used in [5].

For the remainder of the report we shall proceed as follows. In Section 2, we define various types of simple structures for algorithms, a structural graph, and the execution model of an algorithm. In Section 3, we define distributed and centralized processing systems and their respective models. In Section 4, we compare a distributed processing system to a centralized system for various types of service distributions, service disciplines, and various types of structures of algorithms. The results obtained are summarized as three propositions. In addition, we consider examples which indicate which of the possible generalizations of these propositions are not valid. In Section 5, we summarize the results obtained and discuss further work needed to use the analytical model developed in this study for the design of distributed systems.

II. Algorithms and Structural Graphs

As defined in the introduction, a task consists of a set of operations which can be performed on some input data; an algorithm is a collection of tasks with an underlying

structure. A major aspect in the definition of the structure of an algorithm is the precedence relationships (i.e., the partial order of execution) that exist among its tasks.

2.1 Types of Precedence Relationships

We define five basic types of precedence relationships. In the following definitions, the symbols F_a and F_b are used to indicate two distinct tasks, and $\{F_i\}$ and $\{F_j\}$ are used to indicate two distinct sets of tasks which may have some common members.

(a) **Sequential Relationship:** A Sequential relationship $S: F_a \rightarrow F_b$ specifies that for any request to execute these two tasks, task F_a must be completed before task F_b may begin. Following the completion of F_a, task F_b is executed.

(b) **If-then-else Relationship:** An If-then-else relationship $IF: F_a \rightarrow \{F_i\}$ specifies that for any request to execute these tasks, task F_a must be completed before any task in $\{F_i\}$ may begin. Following the completion of F_a, one and only one task in $\{F_i\}$ is selected for execution according to some selection procedure.

(c) **Merge Relationship:** A Merge relationship $M: \{F_i\} \rightarrow F_b$ specifies that for any request to execute these tasks, any task in $\{F_i\}$ must be completed before task F_b may begin. It is assumed here that for any such execution request, only one task in $\{F_i\}$ would be actually executed. Following the completion of this execution, task F_b is executed.

(d) **Fork Relationship:** A Fork relationship $F: F_a \rightarrow \{F_j\}$ specifies that for any request to execute these tasks, task F_a must be completed before tasks in $\{F_i\}$ may begin. Following the completion of F_a, all tasks in $\{F_i\}$ are executed.

(e) **Join Relationship:** A Join relationship $J: \{F_i\} \rightarrow F_b$ specifies that for any request to execute these tasks, all tasks in $\{F_i\}$ must be completed before task F_b may begin. It is assumed here that for any such execution request, all tasks in $\{F_i\}$ would be executed. The order in which the tasks of $\{F_i\}$ are completed is not relevant. Following the completion of tasks $\{F_i\}$, task F_b is executed

A graphical notation used to represent the various types of precedence relationships is given in Figure 1. In this notation, nodes represent tasks, and the directed arcs among nodes (along with the special symbols) represent the order of execution of the tasks.

2.2 Structural Graphs

The structure of an algorithm is represented by a graph based on the graphical notation defined above. Such a graph is called the **structural graph** of the algorithm.

Using the relationships defined above as primitives, one can build some general and complex structural graphs. However, though not readily apparent from their definitions, these primitives may be combined to lead to some "ill-behaved" structures. Figure 2 shows two generic examples of such structures. Figure 2 (a) is an example of a structure in which a task will be executed an infinite number of times for a single request to execute the algorithm. Figure 2 (b) is an example of a structure where the conditions necessary to execute a task are never met, thus leading to a deadlock. In this report, we consider algorithms that are well-behaved; i.e., do not have deadlocks or infinite execution times.

Algorithms in which a task is never executed more than once for a given external execution request are known as **acyclic algorithms**. Examples of acyclic and non-acyclic algorithms are shown in Figure 3. In this report, we restrict algorithms to be acyclic. Our analysis of acyclic algorithms can be extended to include non-acyclic algorithms, but only if the number of times a task is executed for a request to execute the algorithm is deterministic and independent of the input data.

Based on the types of relationships used, structural graphs are divided into three non-overlapping classes: SIFM, PERT, and General. Any structure which has only Sequential, IF-then-else, or Merge types of relationships belongs to the SIFM class. Any structure which has only Sequential, Fork, or Join types of relationships belongs to the PERT class. (This class is named PERT because the structures in it resemble PERT Activity Networks

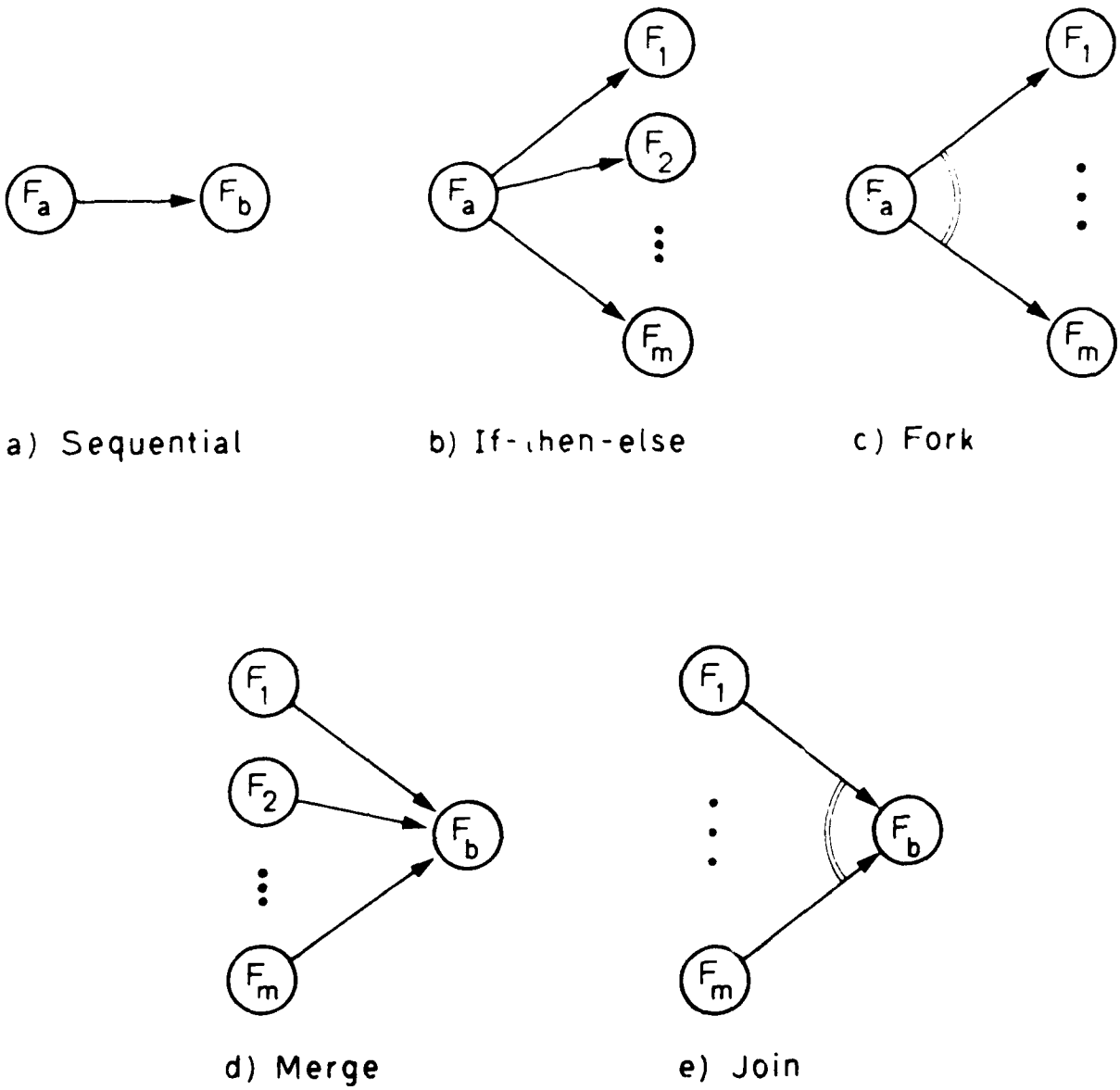


Figure 1. Types of precedence relationships. a) Sequential, b) If-then-else, c) Fork, d) Merge, e) Join.

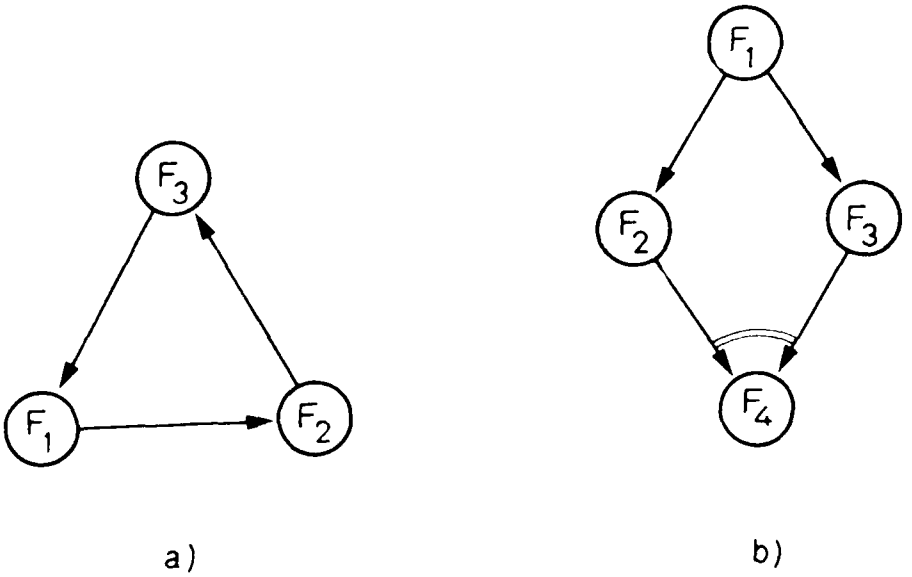


Figure 2. "Ill-behaved" Structural graphs. a) Infinite execution, b) Deadlocks.

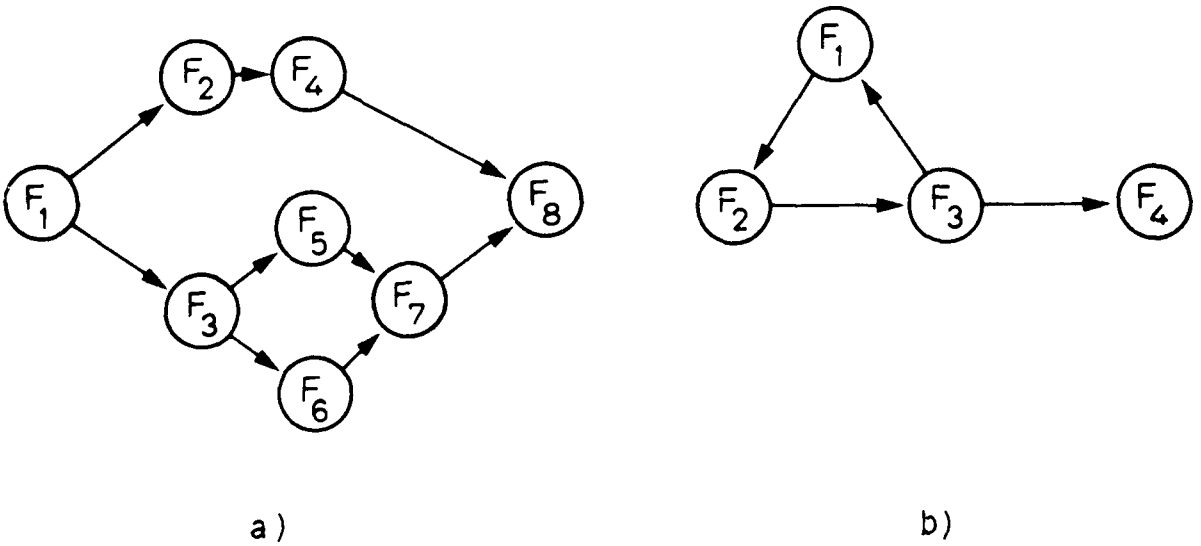


Figure 3. a) An acyclic algorithm. b) A non-cyclic algorithm.

[9].) Any structure that does not fit in these two classes belongs to the General class. Figure 4 shows an example of each class of structure.

Structural graphs can also be characterized according to the type of topology. Three types of topologies can be identified: purely serial (PS), purely parallel (PP), and general (G). Figure 5 shows simple examples of these three types of topologies.

For brevity, the type of a structural graph may be referred to by an ordered list of its attributes; namely, type of precedence relationships, type of topology, and number of tasks. For instance, the structural graph shown in Figure 5 (b) is a SIFM/PP/4 structure of an algorithm.

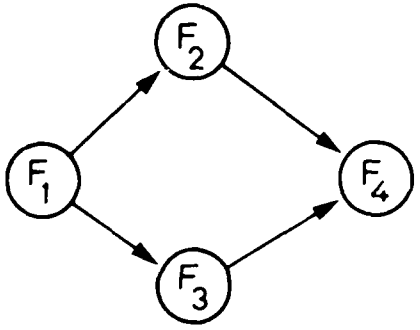
The model of an algorithm and its structure defined here can be extended in many ways. For instance, one may allow the completion of some task to force another ongoing task to complete its execution, or one may allow a task to begin its execution but wait for other tasks to complete their executions before its own execution is completed. We have not considered such a general model because the analysis presented in this report is not adaptable to systems with these characteristics.

2.3 Execution of an Algorithm

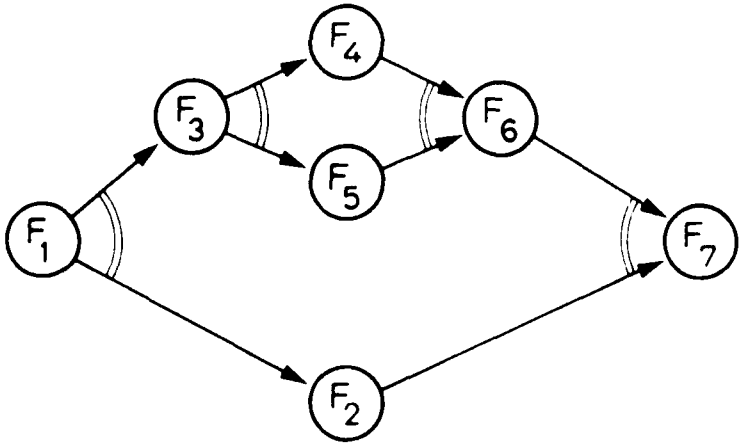
Consider an algorithm A consisting of tasks $\{F_i, i = 1, 2, \ldots, m\}$ and structural graph G. Requests to execute the algorithm arrive from some external source according to some general arrival process $\mathcal{A}$.

In an SIFM type algorithm, an external request to execute the algorithm may be directed to any task in the graph; i.e., the request may be for the execution of the algorithm starting at any particular task. Given the nature of the SIFM type algorithm, no deadlocks will occur, and completion of the algorithm for each input is well-defined.

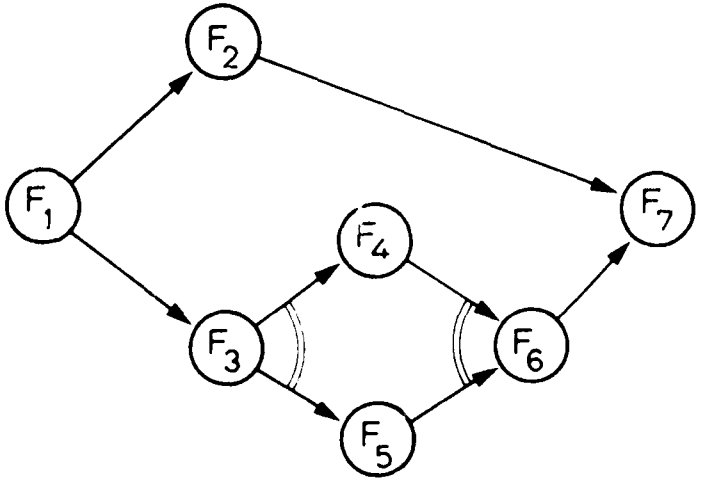
In an arbitrary PERT structure, the above does not hold. The arrival of an external request to a particular task in the graph may lead to a dead-lock, and thus to an ill-defined



a) SIFM

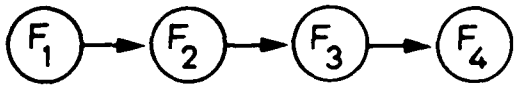


b) Pert

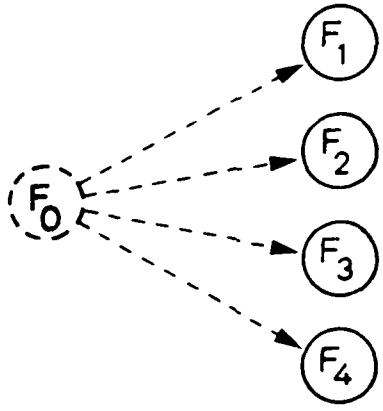


c) General

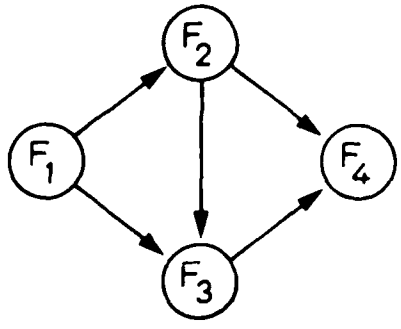
Figure 4. Classes of structures a) SIFM, b) PERT, c) General.



a) SIFM/PS/4



b) SIFM/PP/4



c) SIFM/G/4

Figure 5. Types of topologies of structures. a) Purely Serial (PS), b) Purely Parallel (PP), c) General (G).

notion of algorithm execution. An example of such a situation is depicted in Figure 6, where the arrival of an external request to begin execution at task F_1 cannot be serviced. An external request must be such that both tasks F_1 and F_2 are executed (not necessarily simultaneously). Thus for PERT structures, we shall assume that there exists a single task from which all tasks in the structure can be accessed. (Note that in a PERT structure, the execution of a task results in the execution of all tasks which are downstream of it.) If there is no such no node in the graph, then one can add a "phantom" node F_0 (consisting of no operations) to the structure with the appropriate Fork relationship. (See example in Figure 7.) Note that the phantom node F_0 can also be added even when the single starting node, say F_1, exists, simply by using the sequential relationship $S: F_0 \rightarrow F_1$.

Without loss of generality, we shall also consider that each SIFM structure contains such a phantom node F_0, and an If-then-else relationship $IF: F_0 \rightarrow \{F_i, i = 1, 2, \ldots, m\}$. (See Figure 8.) The selection of the task $F_i, 1 \leq i \leq m$, to follow F_0 is deduced from the arrival process $\mathcal{A}$. The same comments hold for general structures. In general one may have to introduce more than a single phantom node. An example is depicted in Figure 9. Therefore we shall assume that all external requests from process $\mathcal{A}$ are directed to node F_0.

The **execution time** for an external request is the period of the time between the instant of its arrival to the system and the instant at which all tasks associated with that request have completed execution, starting with F_0.

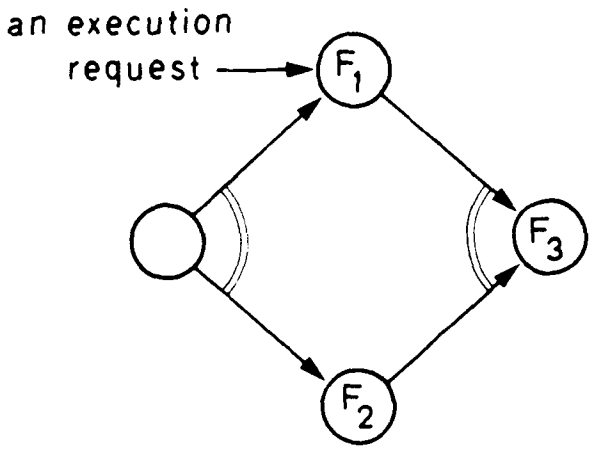


Figure 6. An example of an arrival that cannot be executed.

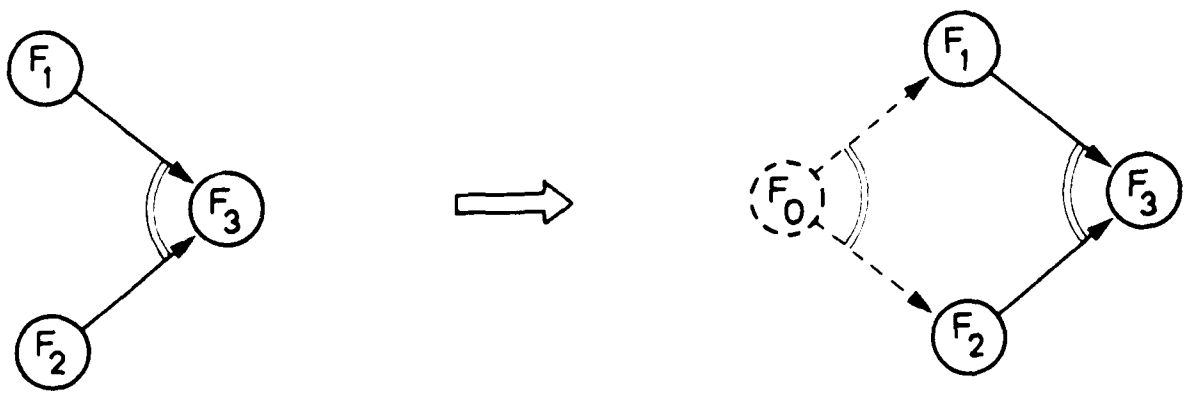


Figure 7. The addition of a "phantom" node, F_0, to a PERT graph.

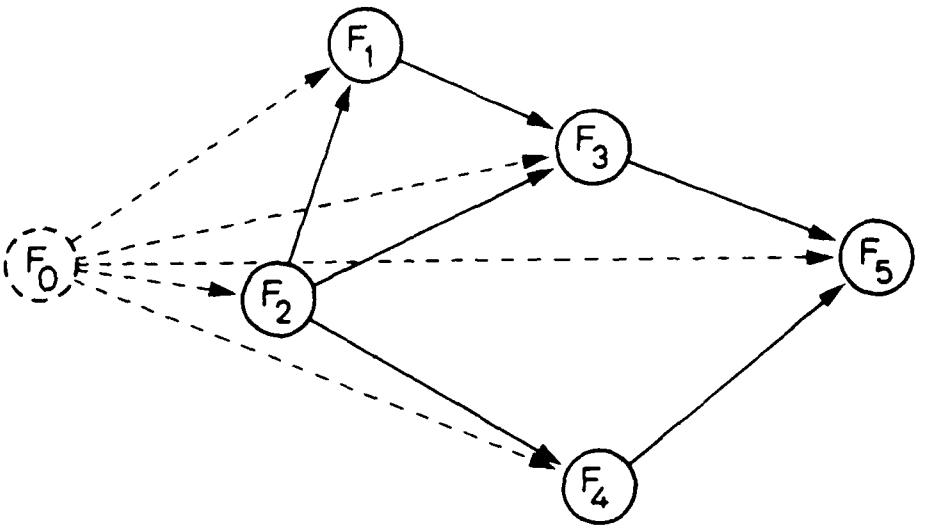


Figure 8. The addition of a "phantom" node, F_0, to a SIFM graph.

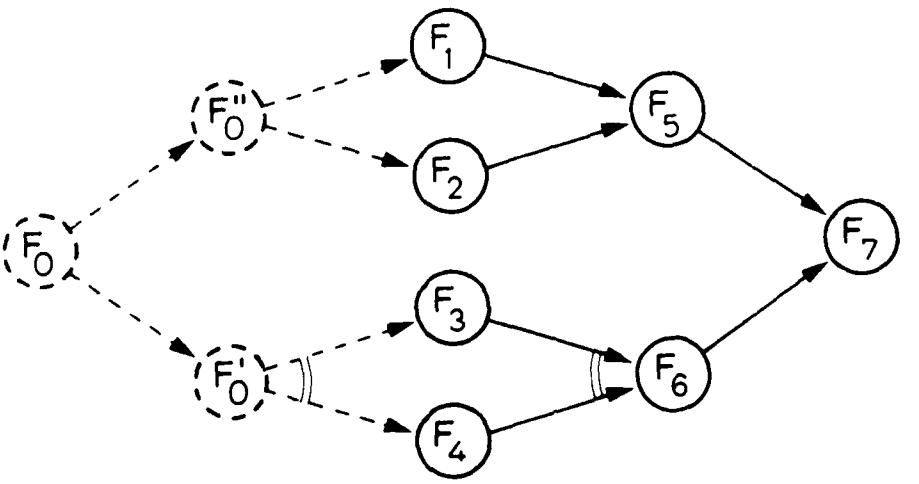


Figure 9. The addition of a "phantom" node to a General graph.

III. Models of Distributed and Centralized Processing Systems

3.1 Model of a Distributed Processing System

Consider an algorithm consisting of a collection of m tasks $\{F_i, i = 1, 2, \ldots, m\}$. A processing system of m processors $\{P_i, i = 1, 2, \ldots, m\}$ is provided such that processor $P_i, 1 \leq i \leq m$, can perform only task F_i. We also consider the existence of a virtual processor P_0 performing task F_0. The processors run asynchronously, but can communicate with each other over a common channel. Each processor is modelled as a single server with infinite waiting room. Once a processor has begun execution of a task, it will run independently and will complete this instance of task execution without any further synchronization among processors. The processor queues any other requests received during its execution of the current task, and acts upon them after completing the execution of this task. Processor P_i performs operations at the rate of C_i operations per second. We assume that the server does not go idle when requests for execution of the task are awaiting, nor does a request leave the processor before receiving full service. Different service disciplines may be considered at the various processors. The service disciplines considered in this report are First-Come-First-Serve (FCFS), Last-Come-First-Serve with Preemptive Resume (LCFS-PR), and Processor-Sharing (PS).

Consider a stream of external requests to execute the algorithm, which arrive according to arrival process A. We number sequentially these requests. Let $E_i, i = 1, 2, \ldots$, denote the i_{th} external request. (Recall that, without loss of generality, we have restricted external requests to arrive to node F_0.) In servicing an external request E_i, requests to perform specific tasks in the algorithm are generated. Define a **job** to be a request to execute a task. Jobs can be uniquely identified by the notation $J_{i,j}$, where i represents the external request, and j the particular task, to which this job corresponds. Note that jobs $\{J_{i,0}, i \geq 1\}$ are

created externally according to the arrival of external execution requests, and there exists one such $J_{i,0}$ for each external request E_i. All other jobs $J_{i,j}, j \neq 0$, are created internally as a result of the completion of some other jobs already in the system, according to the structure of the algorithm. Depending on the types of relationships used in the graph, not all tasks defined in the graph are executed for each execution request. Hence for a given E_i, not all jobs $J_{i,j}, 1 \leq j \leq m$, are created. We define the **execution time** for request E_i, as the time it takes for all jobs $J_{i,*}$ which have been created to be completed, following the creation of $J_{i,0}$.

We now make some simple observations. In an SIFM type graph, one and only one job $J_{i,j}$ for some $j, 1 \leq j \leq m$, exists for a given request E_i at any one time during the execution time of E_i. Hence the execution time of E_i is the time it takes for a specific sequence of jobs $J_{i,0}, J_{i,j_1}, \ldots, J_{i,j_n}, \ldots$ to be executed. Note that this model allows pipelined processing of successive execution requests.

In a PERT type graph, all tasks $J_{i,j}, 1 \leq j \leq m$, must have been created sometime during the execution time of request E_i. It is also possible at any one time to find more than one job $J_{i,j}, 1 \leq j \leq m$, in the system. Note that this model not only allows pipelined processing of successive execution requests, but also allows concurrent processing of tasks to be executed for an execution request.

We now define the arrival process A. We assume that external arrivals of requests (to node F_0) are according to an aggregate **Poisson** process, rate λ requests/second. In the SIFM type graphs, we also consider that following F_0, task $F_i, 1 \leq i \leq m$ is selected with probability q_i. We furthermore assume that the selection of the next task in an If-then-else relationship is random according to a fixed distribution, independent of previous selections.

The number of operations associated with a particular job $J_{i,j}$ is a random variable $\tilde{Z}_{ij}$ which is continuous over the positive real line*. Furthermore, we assume that $\tilde{Z}_{ij}$ is

*This assumption of $\tilde{Z}_{ij}$ being a continuous random variable is reasonable if we assume that the discrete time slot needed to perform each operation is quite small compared to the total service time per task.

independent from all other random variables $\tilde{Z}_{kl}, k \neq i, 1 \leq l \leq m$, and that, for a given j all the random variables $\tilde{Z}_{ij}, i = 1, 2, \ldots$ that are created are independent and identically distributed, with distribution $B_j(x)$. The mean number of operations performed for job $J_{i,j}$ is denoted by $(1/\mu_j)$. The mean number of operations performed for a request is denoted by $(1/\mu)$. This parameter is independent of the system or its model, and is given in terms of $\mu_j, j = 1, 2, \ldots, m$, according to the structure of the algorithm and the arrival process.

3.2 Models of a Centralised System

A single processor is provided which can perform all tasks. The processor is modelled as a single server of capacity C operations per second with infinite waiting room for jobs. On receiving a request to execute the algorithm, the processor will execute in sequence all tasks that are to be performed for this request. While servicing a request, the processor will queue all other requests received.

We consider two alternative models for the service time in a centralized system: a single-stage model and a multi-stage model. In the single-stage model, we view the service time as a single-stage of service with a distribution which is of the same type as that of individual tasks, (assuming all task service distributions $B_j(x), j = 1, 2, \ldots, m$, in the distributed system to be of the same type, e.g. Exponential, Uniform, Hyper-exponential, etc.), and which has a mean service time denoted by $(1/\mu C)$, where C is the capacity of the processor, and μ is the average number of operations that will be executed for a request. This is the model used by Kleinrock [5] with exponential everything. By considering only the average number of operations performed for an execution request in the distributed system, this model ignores the higher moments of the service time (which are determined either by the particular choice of the distribution, as is the case with the exponential distribution, or arbitrarily). We consider the single-stage model to be a first-order approximation.

In the multi-stage model, the service is viewed as a series-parallel arrangement of stages, where each stage corresponds to a service time of a task. Here different stages may have different distributions. For any PERT type graph, the service time is modelled as a single series of stages, and the order of stages in the model is selected arbitrarily but so as to preserve the partial order of execution of the tasks, as specified by the structure of the algorithm. For any SIFM type graph, the service time is modelled as a number of parallel branches of series of stages, where each branch corresponds to a distinct sequence of jobs to be executed for a request. The total number of branches is determined by the structural graph and by the arrival process A. We consider this model to be accurate in the sense that it takes into account the higher moments of the service time distributions of individual tasks.

The definition of execution time in a centralized system is clear since the processor services one request at a time. The performance comparison between the distributed system and the centralized system is based on the average execution time of a request with both systems having the identical arrival process. Figure 10 shows the model of the distributed processing system and the models of the centralized processing system of a SIFM/G/4 algorithm.

3.3 Notation

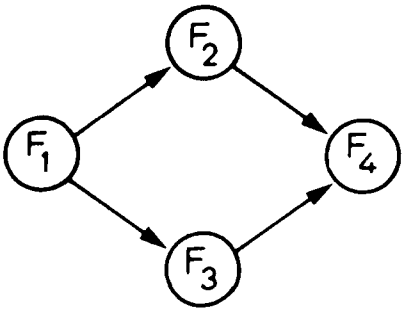
As seen from the previous section, a processing system has many attributes; these are:

a) the type of system model (distributed [d], or centralized single-stage [cs], or centralized multi-stage [cm]),

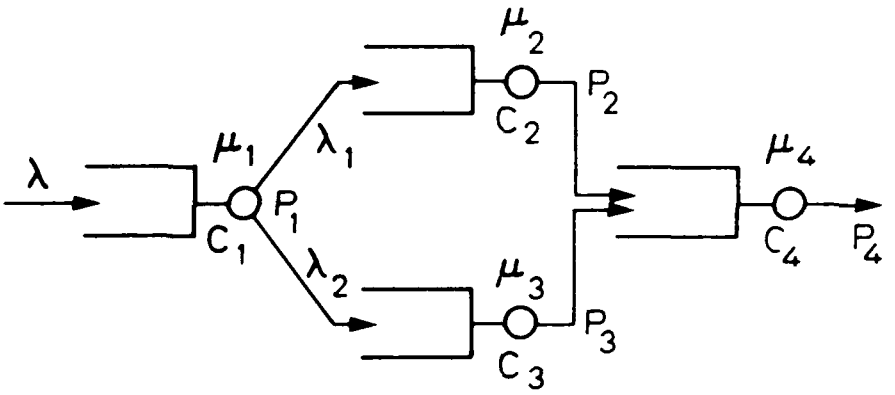
b) the type of precedence relationships (SIFM, PERT, G),

c) the type of topology (PP, PS, G),

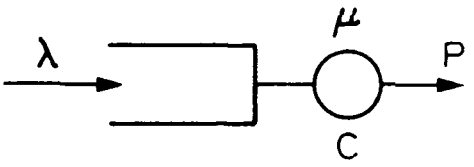
d) the number of tasks in the algorithm (m),



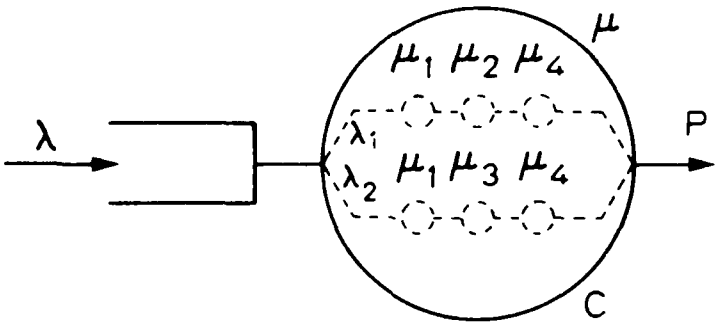
a) A SIFM/G/4 algorithm



b) A d/SIFM/G model



c) A cs/SIFM/G model



d) A cm/SIFM/G model

Figure 10. Models of the distributed and the centralized processing systems for a SIFM/G/4 algorithm.

e) the type of distribution of inter-arrival times of jobs from outside (Poisson [M], Deterministic [D], General [G]),

f) the type of distributions of task services (M, D, G), and

g) the service disciplines within processors (FCFS, LCFS-PR, PS etc.).

In order to concisely refer to processing systems with the same attributes, we list the acronyms corresponding to the attributes in the order listed above; (e.g., d/SIFM/G/m/M/M/FCFS.) The last four of these attributes may be left out at times, if their specification is clear from the context. The average execution time in a system is denoted by a subscripted T, where the subscript consists of the notation that specifies the system. We also use T_d, T_{ce}, and T_{cm}, a more concise notation, when the values of the other system attributes are clear from the context. The ratio of the distributed execution time to the sequential execution time is denoted as $R_{(d,ce)}$ or $R_{(d,cm)}$ depending on the model used for the centralized system. A summary of the notation used in the report is given in Appendix 1.

IV. Distributed versus Centralized Processing Systems

When the resources are limited with the linear constraint, namely $\sum_{i=1}^{m} C_i = C$, Kleinrock [5] states that distributed processing is slower than centralized processing. This was shown to be true for algorithms which consist of a single series of tasks. The centralized system was modelled as a single-stage service model, and the distributions for service times and inter-arrival times were assumed to be exponential.

In this section, we show that while the above result is true even for the more general structures of SIFM and PERT, there are cases of algorithms and models for processing system, namely the multi-stage model for centralized processing systems, where the opposite result is true; that is distributed processing outperforms centralized processing. We

proceed to demonstrate this fact in the following manner. In Section 4.1, we prove the optimality of centralized processing for a class of algorithms, namely SIFM and PERT algorithms, under the single-stage model. The assumptions made in this section are the same as the ones used in [5], and the result obtained here is merely an extension of the result obtained in [5] to more general structures. In section 4.2, while retaining the assumptions made in Section 4.1, we consider the multi-stage model of centralized systems. In this case, the result is shown to be true only for a restricted range of parameters. In this section, we also give an example which shows that there are values for which a distributed system is better than a centralized one. In Section 4.3, we extend the results derived in Section 4.2 to two non-FCFS service disciplines, and to general distributions of service times.

4.1 Centralized Systems with Single-stage Service

The systems considered in this section have the characteristics that (1) the external arrival process is Poisson, (2) each processor uses FCFS service discipline, (3) the service distributions are exponential, (4) the single-stage service model is considered for the centralized system, and (5) the resources are limited such that $\sum_{i=1}^{m} C_i = C$. Under these conditions, the average execution time in a distributed system for a SIFM or a PERT algorithm is higher than that of the centralized system executing the same algorithm. This proposition is true regardless of the structure of the algorithm. This can be stated more concisely as

Proposition 1

(i) $T_{d/SIFM/G/m/M/M/FCFS} \geq T_{cs/SIFM/G/m/M/M/FCFS}$

(ii) $T_{d/PERT/G/m/M/M/FCFS} \geq T_{cs/PERT/G/m/M/M/FCFS}$

Proof:

Part (i)

To prove part (i), let us consider an arbitrary SIFM/G/m algorithm. Our first task is to find the average execution time for the distributed system, T_d. Given the assumptions, the model of the distributed system is analogous to the network of queues model analyzed by Jackson [6]. He showed that the distribution of the number of jobs in the system equals the product of the marginal distribution of the number of jobs at each queue, and the marginal distribution is the same as that of the distribution of jobs in an independent M/M/1 queue. Such networks of queues are referred to as having a product-form solution.

We define the notions of route and class of an external request as follows. In the distributed system, a request generates a sequence of jobs to be executed. We define the **route** of a request as the sequence of processors that execute the sequence of jobs generated by a request. A request is classified according to its route. Let $r, r = 1, 2, \ldots, R$, denote the class of a request, and let a_r denote the route of a request of class r. We shall use $i \in a_r$ to indicate that processor P_i is in route a_r.

Let requests arrive at a rate of λ jobs per second. Given the defined arrival process A, and the random selection of downstream tasks in If-then-else precedence relationship at the "phantom" node, an arrival is of class r with some fixed probability; hence, the arrivals of requests of class r also form a Poisson process. Let the rate of such arrivals be λ_r. Then,

$$\lambda = \sum_{r=1}^{R} \lambda_r \tag{1}$$

Let Y_i be the average execution time of job $J_{*,i}$ at Processor P_i. As shown by Kleinrock [p. 321, 4], the average execution time T_d is expressed in terms of Y_i as

$$T_d = \sum_{i=1}^{m} \frac{\alpha_i}{\lambda} Y_i \tag{2}$$

where

$$\alpha_i = \sum_{\substack{r=1 \\ i \in a(r)}}^{R} \lambda_r \tag{3}$$

is the total rate of arrivals to processor P_i. Because the underlying network of queues model has a product-form solution, Y_i is equal to the average system delay in an independent M/M/1 system that has arrival rate α_i and service rate $\mu_i C_i$. Thus, we have

$$Y_i = \frac{1}{\mu_i C_i - \alpha_i} \tag{4}$$

Eqs. (2), (3), and (4) altogether give us the closed-form solution of the average execution time in the distributed system of any SIFM algorithm.

Our next task is to find the average execution time T_{cs}. When a centralized system is assumed to have a single-stage service facility, this quantity is the average system delay in an M/M/1 system with arrival rate λ and service rate μC, where the parameter μ is given by

$$1/\mu = \sum_{r=1}^{R} \sum_{i \in a(r)} \frac{\lambda_r/\lambda}{\mu_i} = \frac{1}{\lambda} \sum_{i=1}^{m} (\alpha_i/\mu_i) \tag{5}$$

The average execution time, T_{cs}, is thus given by

$$T_{cs} = \frac{1}{\mu C - \lambda} = \frac{\sum_{i=1}^{m} (\alpha_i/\mu_i)}{\lambda [C - \sum_{i=1}^{m} (\alpha_i/\mu_i)]} \tag{6}$$

Given specific values for the various parameters, these equations can be used to calculate and compare execution times of a given algorithm. The proof of part (i) of the

proposition, which is stated for non-specific values of parameters, proceeds by first minimizing over C_i the ratio $R_{(d,ce)}$ of the execution times. The constraints for the minimization problem are $\sum_{i=1}^{m} C_i = C$ and $C_i > 0, i = 1, 2, ..m$. Since C is assumed to be constant, the problem of minimizing the ratio with these constraints is equivalent to the problem of minimizing T_d with the same constraints. The latter minimization problem is the one that we consider next. If all queues are stable, i.e., $\alpha_i/\mu_i C_i < 1$, then the objective function is convex over C_i, as can be shown by twice differentiating Y_i. Hence, the minimization problem has a unique solution.

The minimization problem, solved using the method of Lagrange multiplier, is similar to the capacity assignment (CA) problem considered for computer communication networks [p. 330, 4]. The Lagrangian function is

$$G = T_d + \beta(\sum_{i=1}^{m} C_i - C)$$

where β is the Lagrangian multiplier. Forming the partial derivatives $\frac{\partial Y_i}{\partial C_i}$ for all i, and equating them to zero, we shall get optimal values of C_i in terms of the unknown parameter β. By using the constraint equation $\sum_{i=1}^{m} C_i = C$ to eliminate the unknown parameter β, we obtain the optimal values of C_i as

$$C_i^* = \frac{\alpha_i}{\mu_i} + \frac{C_e\sqrt{\alpha_i/\mu_i}}{\sum_{i=1}^{m}\sqrt{\alpha_i/\mu_i}}$$

where $C_e = C - \sum_{i=1}^{m}(\alpha_i/\mu_i)$. The assumption that all queues are stable guarantees that $C_e \geq 0$, and that the minimization is feasible. From Eq.(6), one finds that the condition $C_e \geq 0$ also guarantees that the single server of the centralized system is stable.

By substituting the optimal value of capacities, C_i^*, in Eqs. (2), (3), and (4), we have

$$\begin{aligned} \min_{C_i} T_d &= \sum_{i=1}^{m} \left(\frac{\alpha_i}{\lambda}\right)\left(\frac{\sum_{i=1}^{m}\sqrt{\alpha_i/\mu_i}}{\mu_i C_e \sqrt{\alpha_i/\mu_i}}\right) \\ &= \frac{[\sum_{i=1}^{m}\sqrt{\alpha_i/\mu_i}]^2}{\lambda C - \lambda \sum_{i=1}^{m}(\alpha_i/\mu_i)} \end{aligned} \tag{7}$$

From Eqs. (6) and (7), the minimum value of the ratio is given as

$$\min_{C_i} R_{(d,ce)} = \frac{[\sum_{i=1}^{m}\sqrt{\alpha_i/\mu_i}]^2}{\sum_{i=1}^{m}(\alpha_i/\mu_i)}$$

Consider the r.h.s. of the above equation. As the expansion of the numerator contains the denominator terms plus some other positive terms, we conclude that the ratio is greater than one for all possible values of parameters C_i, μ_i, and λ_r. This completes the proof of part (i) of the Proposition.

Part (ii)

To prove part (ii) of the Proposition, let us consider an arbitrary PERT/G/m algorithm. In the model of its distributed system, a single request may generate more than one job at a time. This leads to a network of queues model which is not shown to possess a product-form solution; this makes it impossible to find a closed-form solution for T_d. Nonetheless, it is possible to prove part (ii) by making use of the following Lemma.

Lemma 1: From a given PERT/G/m algorithm, we define a new algorithm which has the same tasks, but a purely parallel topology. By abuse of notation, we shall use PERT/PP(G)/m to denote this new algorithm. We claim that

$$T_{d/PERT/G} \geq T_{d/PERT/PP(G)}$$

Proof: Let $\tilde{T}_{d/PERT/G}$ be the execution time of a request to a α/PERT/G system. $\tilde{T}_{d/PERT/G}$ is a random variable which can be expressed as a series of additions and maximizations of individual delay time at processor nodes, say $\{\tilde{Y}_i\}$. In general $\{\tilde{Y}_i\}$ are

dependent random variables. For instance, for the example of Figure 4 (b), we have

$$\tilde{T}_{d/PERT/G} = \max(\tilde{Y}_1 + \tilde{Y}_3 + \max(\tilde{Y}_4, \tilde{Y}_5) + \tilde{Y}_6 + \tilde{Y}_7, \tilde{Y}_1 + \tilde{Y}_2 + \tilde{Y}_7)$$

For a general PERT structure, $\tilde{T}_{d/PERT/G}$ is a function $\Psi(\tilde{Y}_1, \tilde{Y}_2, \ldots, \tilde{Y}_m)$ which depends on the topology of the structure. Let $\tilde{T}_{d/PERT/PP(G)}$ be the execution time of a request to the d/PERT/PP(G). It is given by

$$\tilde{T}_{d/PERT/PP(G)} = \max(\tilde{Y}_1, \tilde{Y}_2, \ldots, \tilde{Y}_m) \tag{8}$$

Then, independent of the function Ψ, one can show that $\tilde{T}_{d/PERT/G} \geq \tilde{T}_{d/PERT/PP(G)}$, and thus

$$T_{d/PERT/G} \stackrel{\Delta}{=} E[\tilde{T}_{d/PERT/G}] \geq E[\tilde{T}_{d/PERT/PP(G)}] \stackrel{\Delta}{=} T_{d/PERT/PP(G)}$$

which proves the lemma.

Next, we shall establish that $T_{d/PERT/PP(G)} \geq T_{cs/PERT/G}$. First of all, we note that from Eq. (8) we have

$$T_{d/PERT/PP(G)} = E[\tilde{T}_{d/PERT/PP(G)}] \geq \max(E[\tilde{Y}_1], E[\tilde{Y}_2], \ldots, E[\tilde{Y}_m]) \tag{9}$$

Since in a d/PERT/PP system, random variables $\tilde{Y}_i$ are independent, and each processor has a rate of λ arrivals per second, the average execution time for the i^{th} processor is

$$E[\tilde{Y}_i] = Y_i = \frac{1}{(\mu_i C_i - \lambda)} \qquad \text{for all } i \tag{10}$$

The r.h.s. of Eq. (9) is minimized over C_i with the constraint $\sum_{i=1}^{m} C_i = C$ when

$$Y_1 = Y_2 = \ldots = Y_m$$

which means that

$$T_{d/PERT/PP(G)} \geq \frac{1}{\mu_1 C_1 - \lambda} = \frac{1}{\mu_2 C_2 - \lambda} = \ldots = \frac{1}{\mu_m C_m - \lambda} \tag{11}$$

From Eq. (11), one finds that the optimal capacities are

$$C_i = \frac{\mu_1}{\mu_i} C_1 \qquad \text{for} \quad i = 2, \ldots, m$$

By substituting these values in the constraint equation, we have

$$\mu_1 C_1 \Big(\sum_{i=1}^{m} \mu_i^{-1} \Big) = C \tag{12}$$

The average execution time in the corresponding centralized system is found easily since the model is an M/M/1 system with arrival rate λ and service rate μC, where the parameter μ is given by

$$\mu^{-1} = \sum_{i=1}^{m} \mu_i^{-1}$$

The average execution time $T_{ce/PERT/G}$ is then given by

$$T_{ce/PERT/G} = \frac{1}{\mu C - \lambda} = \frac{\sum_{i=1}^{m} \mu_i^{-1}}{C - \lambda \sum_{i=1}^{m} \mu_i^{-1}} \tag{13}$$

From Eqs. (11), (12) and (13), we conclude that

$$T_{d/PERT/PP(G)} \geq T_{ce/PERT/G}$$

This inequality along with that of Lemma 1 proves part (ii) of the proposition.

4.2 Centralised Systems with Multi-stage Service

When the service of a centralized system is modelled as a multi-stage service, we show that the result of the comparison between distributed and centralized systems is not always in favor of centralized systems. We shall first establish a result that identifies a range of values of parameters for which centralized systems are always better, and then consider an example where distributed systems are better.

Consider in this section systems which are such that (1) the arrival process is Poisson, (2) all processors use FCFS service discipline, (3) service distributions of all tasks are exponential, (4) the resources are limited such that $\sum_{i=1}^{m} C_i = C$, and (5) the service model in the centralized system is multi-stage. Let C_b^2 be the coefficient of variation of the service time in the centralized system. Under these conditions, the average execution time in a distributed system for a SIFM algorithm is higher than that of a centralized system executing the same algorithm, provided the structure of the algorithm, the arrival distribution and the service distributions are such that $C_b^2 \leq 1$. As we show later in this section, if $C_b^2 > 1$ then the above statement is not true. On the other hand, for a PERT algorithm, centralized systems are always better.

Proposition 2

(i) if $C_b^2 \leq 1$ then $T_{d/SIFM/G/m/M/M/FCFS} \geq T_{cm/SIFM/G/m/M/M/FCFS}$

(ii) $T_{d/PERT/G/m/M/M/FCFS} \geq T_{cm/PERT/G/m/M/M/FCFS}$

Proof:

Part (i)

To prove part (i) of this proposition, let us consider an arbitrary SIFM/G/m algorithm. The average execution time in a distributed system T_d is as in Section 4.1 (See Eqs. 2- 4).

We model the centralized system as an M/G/1 system with the average service time

$(1/\mu C)$, and the coefficient of variation (C_b^2) given as

$$(\mu C)^{-1} = \frac{1}{\lambda C} \sum_{i=1}^{m} (\alpha_i/\mu_i)$$

and

$$C_b^2 = \frac{\sum_{i=1}^{m} (\alpha_i/\lambda\mu_i^2)}{[\sum_{i=1}^{m} (\alpha_i/\lambda\mu_i)]^2} \tag{14}$$

Applying the P-K mean value formula for the average response time of an M/G/1 system [p. 191, 8] we get the average execution time of a centralized system as

$$T_{cm} = \frac{1}{\mu C} + \frac{\rho(1 + C_b^2)}{2\mu C(1 - \rho)}$$

where $\rho \triangleq \lambda/\mu C$ is the utilization factor. By rearranging the terms of the r.h.s. of the above equation, we have

$$T_{cm} = (\frac{1}{\mu C - \lambda})(1 + \frac{\rho}{2}(C_b^2 - 1))$$

Since the μ and λ are the same for the single-stage model as for the multi-stage model, we use Eq. (6) to derive the following relation between average execution times of these two models.

$$T_{cm} = T_{cs}(1 + \frac{\rho}{2}(C_b^2 - 1)) \tag{15}$$

The utilization factor ρ is always non-negative. Hence, if $C_b^2 \leq 1$ then $T_{cs} \geq T_{cm}$. This inequality and part (i) of Proposition 1 proves part (i) of this proposition.

Part (ii)

To prove part (ii), consider an arbitrary PERT/G/m algorithm. The average execution time in a distributed system T_d has not changed from that of Section 4.1 (See Eqs. 8- 11).

For any PERT algorithm, the multi-stage service model is hyper-exponential, i.e., a single series of exponential stages. The average service time $(1/\mu C)$ is

$$\frac{1}{\mu C} = \sum_{i=1}^{m} \frac{1}{\mu_i C}$$

Since the service time distribution is hyper-exponential, the coefficient of variation (C_b^2) is never greater than one.

The μ and λ are the same for the single-stage and for the multi-stage model. Hence, the relationship between T_{cm} and T_{cs} as given by Eq. (15) is valid. And, since ρ is always non-negative, and C_b^2 is never greater than one, we have $T_{cm} \leq T_{cs}$ for any PERT algorithm. This inequality and part (ii) of Proposition 1 proves part (ii) of this proposition.
QED

We now investigate the relative performance of the systems when $C_b^2 \geq 1$ by considering an example of a SIFM/PP/2 algorithm. The observations made for this case can be extrapolated to a SIFM/PP/m algorithm. Note that when $\mu_i = \mu, \quad i = 1, 2$, the coefficient of variation C_b^2 is one. Hence, from Eq. (15) it is obvious that centralized systems are always better if the μ_i's are equal. This is the result that was shown by Kleinrock [4].

Our objective is to explore the case where $\mu_1 \neq \mu_2$, and hence $C_b^2 > 1$. Using Eqs. (2), (3), and (4) to get T_d, and Eqs. (6), (14), and (15) to get T_{cm}, we get the ratio $R_{d,cm}$. Minimizing the ratio over C_i with the constraint $C_1 + C_2 = C$ gives

$$R^*_{(d,cm)} \triangleq \min_{C_i} R_{(d,cm)} = \frac{[\sum_{i=1}^{2} \sqrt{\lambda_i/\lambda\mu_i}]^2}{[\sum_{i=1}^{2}(\lambda_i/\lambda\mu_i)][1 + (\rho/2)(C_b^2 - 1)]} \tag{16}$$

where

$$C_b^2 = \frac{\sum_{i=1}^{2}(\lambda_i/\lambda\mu_i^2)}{[\sum_{i=1}^{2}(\lambda_i/\lambda\mu_i)]^2} \tag{17}$$

and

$$\rho \triangleq \lambda/\mu C = (1/C)\sum_{i=1}^{2}(\lambda_i/\lambda\mu_i)$$

Now, we shall show that for some values of the parameters, $R_{(d,cm)} < 1$. Note that as long as $\mu_1 \neq \mu_2$, $C_b^2 > 1$. Let us select $\lambda_1/\lambda = \epsilon, \mu_1 = \epsilon^2$, and $\mu_2 = 1$. For $\epsilon << 1$, this assignment of values corresponds to a situation where a large number of requests require very little work to be done, i.e., arrive to processor P_1. Substituting the above values in Eqs. (16) and (17), we can easily show that as $\epsilon \rightarrow 0$, and as $\rho \rightarrow 1, R^*_{(d,cm)} \rightarrow 0$. In Figure 11, we plot $R^*_{(d,cm)}$ versus ϵ, i.e., λ_1/λ, for $\rho = 0.1, 0.5,$ and 0.9. These curves clearly show that in this example there are many values of λ_1/λ for which the distributed system is better than the centralized system. Hence we conclude that for the given assumptions, part (i) of Proposition 2 cannot be extended to all SIFM algorithms.

4.3 Non-Exponential Service Times

In the previous sections, the service received by a job at a processor in the distributed system was assumed to be exponentially distributed. The assumption may be appropriate for tasks which involve data-dependent iterative operations. But, there are tasks which involve a constant number of operations. Hence, we consider in this section the cases where $B_i(x)$ have non-exponential distributions.

The first case that we consider is that of a system which uses either LCFS-PR or PS service discipline. We consider these disciplines first because the model of the distributed system of a SIFM/G/m algorithm has a product-form solution, and hence is easy to analyze.

The systems that we consider in this section are such that (1) the arrival process is Poisson (2) processors uses either LCFS-PR or PS service discipline (3) service time

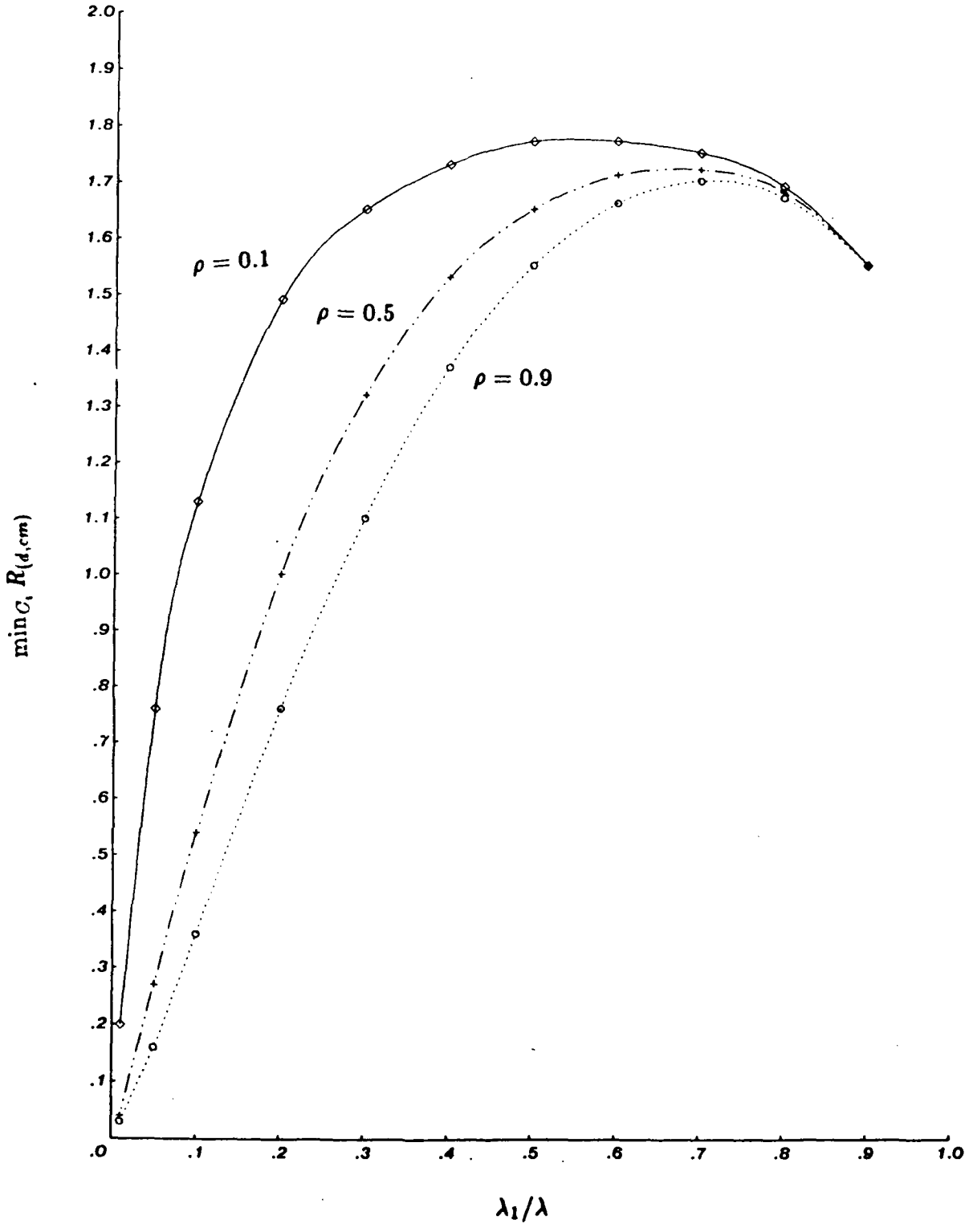


Figure 11. A performance comparison of the distributed and the centralized systems of a SIFM/PP/2 algorithm with $\mu_1 = 1$, $\mu_2 = (\lambda_1/\lambda)^2$, and $\lambda_1 + \lambda_2 = \lambda$.

distributions are general, (4) resources are limited such that $\sum_{i=1}^{m} C_i = C$, and (5) multi-stage service model is used. With these assumptions, the average execution time for a distributed system of a SIFM or a PERT algorithm is higher than that of a centralized system executing the same algorithm, regardless of the actual distributions or the algorithm structure.

Proposition 3 Using **X** service discipline to mean that processors uses either LCFS-PR or PS service discipline, we can state the above proposition more concisely as

(i) $T_{d/SIFM/G/m/M/G/X} \geq T_{cm/SIFM/G/m/M/G/X}$

(ii) $T_{d/PERT/G/m/M/G/X} \geq T_{cm/PERT/G/m/M/G/X}$

The proof of this proposition hinges on two facts. The first fact to note is that the network of queues model of a d/SIFM/G/m/M/G/X system has a product-form solution [7]. Hence, the average execution time at a processor, i.e. T_i, is given as if the ith queue is running independently with Poisson arrivals at a rate α_i. The second fact to note is that the average system delay of an M/G/1 system with either LCFS-PR or PS service discipline depends only on the first moment of its service time distribution [8].

Proof:

Part (i)

To prove part (i), let us consider a SIFM/G/m algorithm. Since the model of the distributed system has a product-form solution, the i_{th} processor behaves as an independent M/G/1 system with either a LCFS-PR or a PS service discipline, the arrival rate of α_i, and a general service distribution with the first moment denoted by $(1/\mu_i C_i)$. Let Y_i be the average execution time of a job at the i_{th} processor. For such a model, it has been shown in [8] that

$$Y_i = \frac{1}{\mu_i C_i - \alpha_i}$$

As this Eq. is identical to Eq. (4), the rest of the proof is similar to that of part (i) of Proposition 1.

Part (ii)

To prove part (ii), let us consider a PERT/G/m algorithm. First of all, we note that Lemma 1 is valid for any service time distribution. Hence, we have

$$T_{d/PERT/G} \geq T_{d/PERT/PP(G)}$$

In the d/PERT/PP(G) system, each processor is modelled as an M/G/1 system with either a LCFS-PR or a PS service discipline, arrival rate λ, and a general service time distribution with the first moment denoted by $(1/\mu_i C_i)$. Let Y_i be the average execution time of a job at the i_{th} processor. As noted above for this system, Y_i is given by

$$Y_i = \frac{1}{\mu_i C_i - \lambda}$$

As this Eq. is identical to Eq. (10), the rest of the proof is similar to that of part (ii) of Proposition 1.

QED

It is difficult to obtain a similar result for a SIFM/G/m algorithm with general service distributions and FCFS service disciplines at processors, since the model of its distributed system does not have a product-form solution. But, it is possible to find closed-form solutions for some particular systems, e.g. d/SIFM/PP systems. In the following, we shall consider a specific case, namely, a SIFM/PP/2/M/D/FCFS system.

Both processors of the distributed system are modelled as M/D/1 systems with FCFS service discipline. The arrival rate is $\lambda_i, i = 1,2$, and the service rate is $\mu_i C_i, i = 1,2$ at processor $P_i, i = 1,2$. Hence, the average execution time (Y_i) at the i_{th} processor is

$$Y_i = \frac{2\mu_i C_i - \lambda_i}{\mu_i C_i (\mu_i C_i - \lambda_i)}$$

and the average execution time (T_d) of a request is

$$T_d \triangleq \sum_{i=1}^{2}(\frac{\lambda_i}{\lambda})Y_i = \sum_{i=1}^{2}\frac{\lambda_i(2\mu_i C_i - \lambda_i)}{\lambda\mu_i C_i(\mu_i C_i - \lambda_i)}$$

The processor of the centralized system is also modelled as an M/D/1 system with FCFS service discipline. The arrival rate is λ and the service rate is μC, where the parameter μ is given by

$$\mu^{-1} = \sum_{i=1}^{2}\lambda_i/\lambda\mu_i$$

The average execution time (T_{cm}) is

$$T_{cm} = \frac{2\mu C - \lambda}{\mu C(\mu C - \lambda)}$$

Thus, the ratio $R_{(d,cm)}$ is

$$R_{(d,cm)} \triangleq \frac{T_d}{T_{cm}} = \frac{\sum_{i=1}^{2}(\lambda_i/\mu_i C_i)(2\mu_i C_i - \lambda_i)/(\mu_i C_i - \lambda_i)}{(\lambda/\mu C)(2\mu C - \lambda)/(\mu C - \lambda)}$$

The number of independent parameters in the above equation can be reduced with the definition of a new parameter, $K \triangleq (\lambda_1/\mu_1)/(\lambda_2/\mu_2)$, which is called the design ratio. The ratio $R_{(d,cm)}$, expressed in terms of the parameters K, ρ, and $C_i, i = 1, 2$, is

$$R_{(d,cm)} = \frac{\sum_{i=1}^{2}[a_i(2C_i - \rho a_i)/C_i(C_i - \rho a_i)]}{(2-\rho)/(1-\rho)} \tag{18}$$

where $a_1 = K/1 + K$, and $a_2 = 1/1 + K$.

We shall show that for this case the centralized system is always better than the distributed system. In order to show this we shall first consider the minimization of the ratio over C_i with the constraint, $C_1 + C_2 = C$. Let $F(K, \rho, C_1, C_2)$ denote the numerator of the r.h.s. of Eq. (18). The above minimization problem is equivalent to the problem of

minimizing $F(.)$ over C_i for the same constraint. The function $F(.)$ is convex over C_i, as can be shown by twice differentiating it with respect to C_i. Using the Lagrangian method gives a set of fourth-degree polynomial equations that, in theory, can be simultaneously solved. In practice, the solution is too unwieldy. Hence, we chose to use the convex programming method. We calculated the minimum of $F(.)$ (and $R_{(d,cm)}$) over C_i for a given ρ and K using a gradient-search type of algorithm. In Figure 12, we show the minimum values of $R_{(d,cm)}$ thus obtained for various values of the design ratio K and ρ. There is very little variation observed as ρ varies in the range 0 to 1. For any value of ρ, as K varies in the range $(0, \infty)$ the minimum value of the ratio is always greater than one. Also, using Eq. (18), it can be shown that as $K \rightarrow 0$ or $K \rightarrow \infty$, $R_{(d,cm)} \rightarrow 1$. Thus, for this specific example the distributed system is better than the centralized system. Given this result about a SIFM/PP/2/M/D/FCFS system, it is easy to show that the same result is true for a SIFM/PP/m/M/D/FCFS system.

$K = \frac{\lambda_1/\mu_1}{\lambda_2/\mu_2}$	$\min_{C_i} R_{d,cm}$		
	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
0.1	1.56	1.59	1.60
0.5	1.94	1.95	1.95
1	2.00	2.00	2.00
5	1.75	1.76	1.76
10	1.58	1.59	1.60
50	1.28	1.29	1.31
100	1.20	1.21	1.23
1000	1.06	1.07	1.08

Figure 12. A SIFM/PP/2/M/D/FCFS system.

V. Conclusions

In this report, we have considered the relative performance of distributed processing and centralized processing of an algorithm. The algorithms were abstracted as consisting of a number of tasks with precedence constraints among them. The types of precedence relationships were such as to allow pipelined processing of various requests, and concurrent processing of tasks for a request in a distributed processing system. Our main assumption about the processing systems used was that the processing capacity of a centralized system is the sum of capacities of processors in a distributed system.

In this report, we have shown that, given the constraint on the capacities of processors, the centralized system does not always outperform the distributed system. The result differs from the one obtained in [5]. The difference is not due to a general model of relationship among tasks proposed here, but is due to our use of an exact model for a centralized system, namely the multi-stage model. We have also shown the range of parameter values for which distributed systems are always worse, and have given examples for which a distributed system is better. Even though we have found specific cases of algorithms for which the distributed processing outperforms centralized processing, in most cases the opposite result holds true. Hence, a good heuristic rule of design is to use a centralized processing system as long as the technological considerations do not prevent the manufacturing of the processor of the required capacity, and as long as the linear cost constraint is applicable.

The approach used here in modelling distributed processing is a new one. The question that arises in this context is whether one could use this approach for achieving a general theory of distributed processing that will help in designing distributed systems. The answer is not very encouraging primarily because of the following short-comings. The major problem is the inability to analyze exactly the algorithms that use Fork and Join relationships, and the algorithms that have general service distributions (and FCFS disciplines

for each processor of the systems). Another problem with the model is that it ignores the impact of communications among tasks. We hope to carry out further research to deal with some of these issues.

Appendix I

The following is a useful summary of the notation used in the report.

P_i	a processor in a distributed system
F_i	task assigned to processor P_i
E_i	a request to execute a algorithm
$J_{i,j}$	A job to execute task F_j for a request E_i
μ_i	mean number of operations performed for a job $J_{*,i}$
C_i	total capacity (in operation/sec) of processor P_i
m	the total number of tasks in a algorithm
r	a class of a request
R	the total number of classes
a_r	route of a request of class r
λ	external arrival rate in requests/second
λ_r	external arrival rate of requests of class r
α_i	total arrival rate to processor P_i
Z_r	mean execution time for a request of class r
Y_i	mean execution time of a job $J_{*,i}$ at Processor P_i
C	total capacity of the processor in a centralized system
μ	mean number of operations performed for a request in a centralized system
C_b^2	coefficient of variation of service time in a centralized system
T_d	mean execution time for a request in a distributed system
T_{cs}	mean execution time for a request in a centralized system with single-stage service
T_{cm}	mean execution time for a request in a centralized system with multi-stage service
$R_{(d,cs)}$	$\triangleq T_d/T_{cs}$
$R_{(d,cm)}$	$\triangleq T_d/T_{cm}$

References

[1] Morse, P. M., *Queues, Inventories and Maintenance*, John Wiley (New York), 1958.

[2] Stidham, S. Jr., "On the Optimality of Single-Server Queueing Systems," Operations Research, 18, No. 4, 708-732 (1970).

[3] Brumelle, S. L., "Some Inequalities for Parallel Service Queues," Operations Research, 19, 402-413 (1971).

[4] Kleinrock, L., *Queueing Systems Volume 2: Computer Applications*, John Wiley (New York), pp. 272-290 (1976).

[5] Kleinrock, L., "On the Theory of Distributed Processing," (1984).

[6] Jackson, J. R.,"Jobshop-like Queueing systems," Management Science, 10, 131-142 (1963).

[7] Baskett, F., K. M. Chandy, R. R. Muntz, and F. Palacious-Gomez, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," Journal of the Association for Computing Machinery, 22, 248-260 (1975).

[8] Kleinrock, L., *Queueing Systems Volume 1: Theory*, John Wiley (New York), 1975

[9] Elmaghraby, S., *Activity Networks: Project Planning and Control by Network Models*, John Wiley (New York), 1977